# 01_homework_dt_and_knn

October 29, 2017

```
In [ ]: import numpy as np
        from sklearn import datasets, model_selection
        import matplotlib.pyplot as plt
        %matplotlib inline
```

## 1 Programming assignment 1: Decision Trees

```
In [284]: data = np.genfromtxt(fname='01_homework_dataset.csv', skip_header=1, delimiter=',')
          X_train, y_train = data[:, :3], data[:, 3]

In [285]: X_train

Out[285]: array([[ 5.5,  0.5,  4.5],
                 [ 7.4,  1.1,  3.6],
                 [ 5.9,  0.2,  3.4],
                 [ 9.9,  0.1,  0.8],
                 [ 6.9, -0.1,  0.6],
                 [ 6.8, -0.3,  5.1],
                 [ 4.1,  0.3,  5.1],
                 [ 1.3, -0.2,  1.8],
                 [ 4.5,  0.4,  2. ],
                 [ 0.5,  0. ,  2.3],
                 [ 5.9, -0.1,  4.4],
                 [ 9.3, -0.2,  3.2],
                 [ 1. ,  0.1,  2.8],
                 [ 0.4,  0.1,  4.3],
                 [ 2.7, -0.5,  4.2]])

In [286]: y_train

Out[286]: array([ 2.,  0.,  2.,  0.,  2.,  2.,  1.,  1.,  0.,  1.,  0.,  0.,  1.,
                  1.,  1.])
```

### 1.1 Helper methods

```
In [287]: def get_gini_index(x, y, num_classes=3):
              sigma = 0.0
              for i in range(num_classes):
```

```
            sigma += _get_pi_c(x, y, i, num_classes)**2
        return 1 - sigma
```

In [288]:
```
def _plot(x, y, f):
    ig = []
    splits = x[:, f]
    min_s, max_s = np.min(splits), np.max(splits)
    for s in splits:
        if s == min_s or s == max_s:
            ig.append(0)
        else:
            ig.append(get_info_gain(x, y, s, f))
    plt.plot(splits, ig, '.')
    plt.show()
```

In [289]:
```
def _get_pi_c(x, y, c, num_classes=3):
    denom = np.sum([y == c])
    nom = 0.0
    for i in range(num_classes):
        nom += np.sum([y == i])
    return denom / len(y)
```

In [290]:
```
def get_info_gain(x, y, s, f):
    num_train_samples = x.shape[0]
    delta_s_t = 0.0
    idx_L = np.where(x[:, f] <= s)
    idx_R = np.where(x[:, f] > s)
    p_L = len(idx_L) / num_train_samples
    p_R = len(idx_R) / num_train_samples
    i_G_t_L = get_gini_index(x[idx_L, :], y[idx_L])
    i_G_t_R = get_gini_index(x[idx_R, :], y[idx_R])

    delta_s_t = get_gini_index(x, y) - (p_L * i_G_t_L + p_R * i_G_t_R)

    return delta_s_t
```

In [291]:
```
def get_best_split(x, y, f):
    best_info_gain = 0.0
    best_split = 0.0
    best_idx = -1
    splits = x[:,f]
    min_s, max_s = np.min(splits), np.max(splits)
    for i, s in enumerate(splits):
        if s == min_s or s == max_s:
            continue
        else:
            ig = get_info_gain(x, y, s, f)
            if ig >= best_info_gain:
                best_info_gain = ig
```

```
                    best_idx = i
                    best_split = s
        return best_info_gain, best_split, best_idx
```

```
In [292]: def get_dataset_children(f, s, x, y):
            idx_l = np.where(x[:, f] <= s)[0]
            idx_r = np.where(x[:, f] > s)[0]
            return x[idx_l,:], y[idx_l], x[idx_r,:], y[idx_r]
```

### 1.1.1 Problem 1

```
In [ ]:      x0 <= 4.1, Gini(5,6,4) = 0.65
                        /     \
        Gini(0,6,0) = 0.0   x0 <= 6.9, Gini(5,0,4) = 0.49
                                      /     \
                        Gini(2,0,4) = 0.61  Gini(3,0,0) = 0.0
```

### 1.1.2 Problem 2

$$y_a = 1, p(c = y_a | x_a, T) = 1.0$$
$$y_b = 2, p(c = y_b | x_b, T) = 0.66$$

# 2 Programming assignment 1: k-Nearest Neighbors classification

## 2.1 Introduction

For those of you new to Python, there are lots of tutorials online, just pick whichever you like best :)

If you never worked with Numpy or Jupyter before, you can check out these guides * https://docs.scipy.org/doc/numpy-dev/user/quickstart.html * http://jupyter.readthedocs.io/en/latest/

## 2.2 Your task

In this notebook code to perform k-NN classification is provided. However, some functions are incomplete. Your task is to fill in the missing code and run the entire notebook.

In the beginning of every function there is docstring, which specifies the format of input and output. Write your code in a way that adheres to it. You may only use plain python and `numpy` functions (i.e. no scikit-learn classifiers).

Once you complete the assignments, export the entire notebook as PDF using nbconvert and attach it to your homework solutions. On a Linux machine you can simply use `pdfunite`, there are similar tools for other platforms too. You can only upload a single PDF file to Moodle.

## 2.3 Load dataset

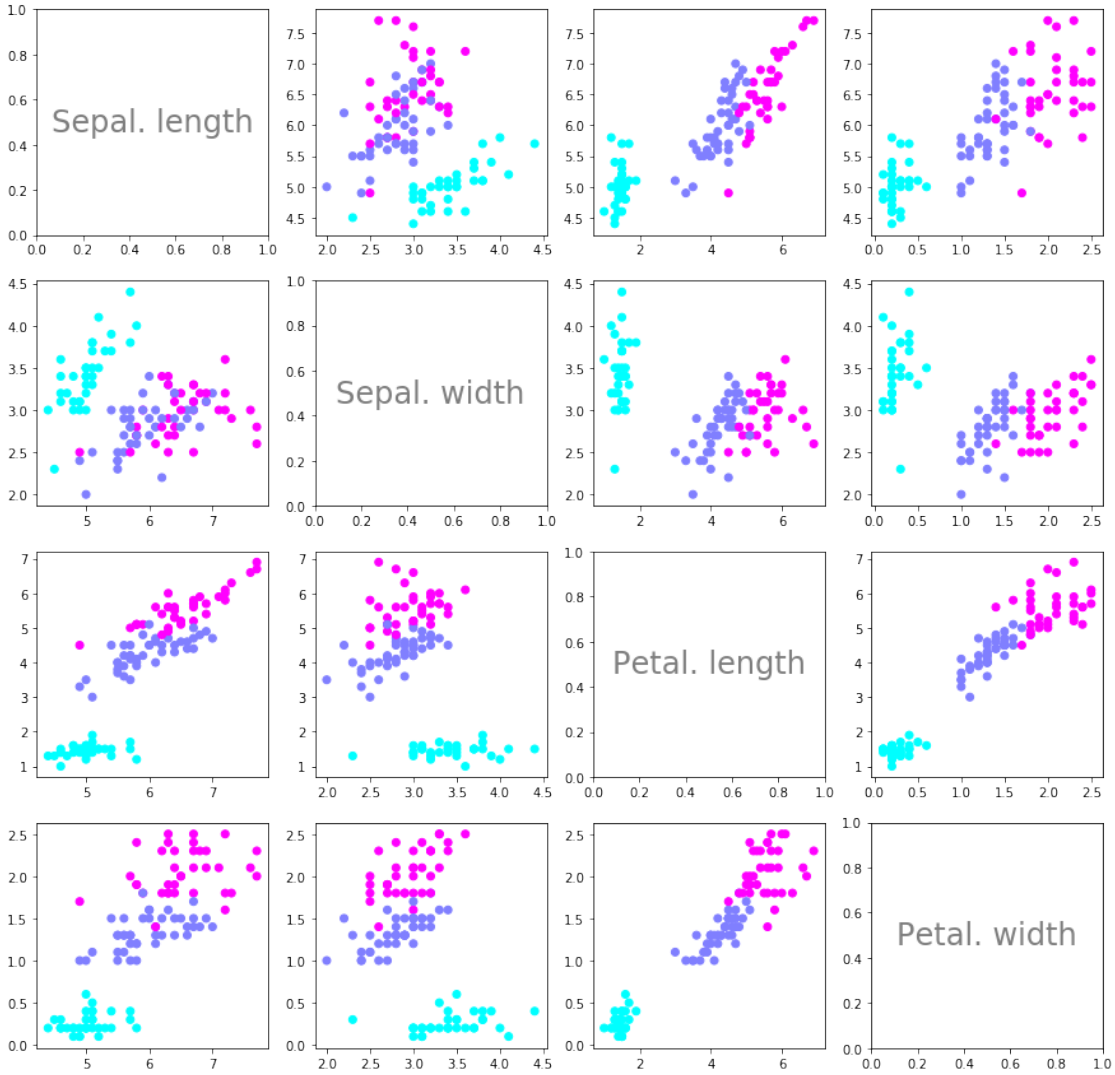
The iris data set (https://en.wikipedia.org/wiki/Iris_flower_data_set) is loaded and split into train and test parts by the function `load_dataset`.

```
In [79]: def load_dataset(split):
             """Load and split the dataset into training and test parts.

             Parameters
             ----------
             split : float in range (0, 1)
                 Fraction of the data used for training.

             Returns
             -------
             X_train : array, shape (N_train, 4)
                 Training features.
             y_train : array, shape (N_train)
                 Training labels.
             X_test : array, shape (N_test, 4)
                 Test features.
             y_test : array, shape (N_test)
                 Test labels.
             """
             dataset = datasets.load_iris()
             X, y = dataset['data'], dataset['target']
             X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, random_s
             return X_train, X_test, y_train, y_test

In [80]: # prepare data
         split = 0.67
         X_train, X_test, y_train, y_test = load_dataset(split)
```

### 2.4 Plot dataset

Since the data has 4 features, 16 scatterplots (4x4) are plotted showing the dependencies between each pair of features.

```
In [81]: f, axes = plt.subplots(4, 4,figsize=(15, 15))
         for i in range(4):
             for j in range(4):
                 if j == 0 and i == 0:
                     axes[i,j].text(0.5, 0.5, 'Sepal. length', ha='center', va='center', size=
                 elif j == 1 and i == 1:
                     axes[i,j].text(0.5, 0.5, 'Sepal. width', ha='center', va='center', size=2
                 elif j == 2 and i == 2:
                     axes[i,j].text(0.5, 0.5, 'Petal. length', ha='center', va='center', size=
                 elif j == 3 and i == 3:
                     axes[i,j].text(0.5, 0.5, 'Petal. width', ha='center', va='center', size=2
                 else:
                     axes[i,j].scatter(X_train[:,j],X_train[:,i], c=y_train, cmap=plt.cm.cool)
```




## 2.5 Task 1: Euclidean distance

Compute Euclidean distance between two data points.

```
In [117]: def euclidean_distance(x1, x2):
              """Compute Euclidean distance between two data points.

              Parameters
              ----------
              x1 : array, shape (4)
                  First data point.
              x2 : array, shape (4)
                  Second data point.
```

```
    Returns
    -------
    distance : float
        Euclidean distance between x1 and x2.
    """
    diff = (x1 - x2)
    return np.sqrt(np.sum(diff**2))
```

### 2.6 Task 2: get k nearest neighbors' labels

Get the labels of the *k* nearest neighbors of the datapoint *x_new*.

```
In [118]: def get_neighbors_labels(X_train, y_train, x_new, k):
    """Get the labels of the k nearest neighbors of the datapoint x_new.

    Parameters
    ----------
    X_train : array, shape (N_train, 4)
        Training features.
    y_train : array, shape (N_train)
        Training labels.
    x_new : array, shape (4)
        Data point for which the neighbors have to be found.
    k : int
        Number of neighbors to return.

    Returns
    -------
    neighbors_labels : array, shape (k)
        Array containing the labels of the k nearest neighbors.
    """
    num_train_samples = X_train.shape[0]
    dist = np.zeros((num_train_samples,))
    for i in range(num_train_samples):
        dist[i] = euclidean_distance(X_train[i], x_new)
    k_nearest_idx = np.argsort(dist)[:k]
    return y_train[k_nearest_idx]
```

### 2.7 Task 3: get the majority label

For the previously computed labels of the *k* nearest neighbors, compute the actual response. I.e. give back the class of the majority of nearest neighbors. Think about how a tie is handled by your solution.

```
In [119]: def get_response(neighbors, num_classes=3):
    """Predict label given the set of neighbors.

    Parameters
```

```
    ----------
    neighbors_labels : array, shape (k)
        Array containing the labels of the k nearest neighbors.
    num_classes : int
        Number of classes in the dataset.

    Returns
    -------
    y : int
        Majority class among the neighbors.
    """
    # TODO
    num_neighbors = neighbors.shape[0]
    class_votes = np.zeros(num_classes)
    for i in range(num_classes):
        class_votes[i] = np.sum(np.where(neighbors == i)[0])
    return np.argmax(class_votes)
```

### 2.8 For Problem 5: get the mean label

For the previously computed labels of the $k$ nearest neighbors, compute the actual response. I.e. give back the class of the mean of nearest neighbors.

```
In [120]: def get_response_reg(neighbors):
              """Predict label given the set of neighbors.

              Parameters
              ----------
              neighbors_labels : array, shape (k)
                  Array containing the labels of the k nearest neighbors.
              num_classes : int
                  Number of classes in the dataset.

              Returns
              -------
              y : int
                  Mean value among the neighbors.
              """
              # TODOs
              return np.mean(neighbors)
```

### 2.9 Task 4: compute accuracy

Compute the accuracy of the generated predictions.

```
In [121]: def compute_accuracy(y_pred, y_test):
              """Compute accuracy of prediction.
```

```
        Parameters
        ----------
        y_pred : array, shape (N_test)
            Predicted labels.
        y_test : array, shape (N_test)
            True labels.
        """
        TP = np.sum(np.equal(y_pred, y_test))
        return TP / y_test.shape[0]
```

In [122]:
```
# This function is given, nothing to do here.
def predict(X_train, y_train, X_test, k):
    """Generate predictions for all points in the test set.

    Parameters
    ----------
    X_train : array, shape (N_train, 4)
        Training features.
    y_train : array, shape (N_train)
        Training labels.
    X_test : array, shape (N_test, 4)
        Test features.
    k : int
        Number of neighbors to consider.

    Returns
    -------
    y_pred : array, shape (N_test)
        Predictions for the test data.
    """
    y_pred = []
    for x_new in X_test:
        neighbors = get_neighbors_labels(X_train, y_train, x_new, k)
        y_pred.append(get_response(neighbors))
    return y_pred
```

In [123]:
```
def predict_reg(X_train, y_train, X_test, k):
    """Generate predictions for all points in the test set.

    Parameters
    ----------
    X_train : array, shape (N_train, 4)
        Training features.
    y_train : array, shape (N_train)
        Training labels.
    X_test : array, shape (N_test, 4)
        Test features.
    k : int
```

```
        Number of neighbors to consider.

    Returns
    -------
    y_pred : array, shape (N_test)
        Predictions for the test data.
    """
    y_pred = []
    for x_new in X_test:
        neighbors = get_neighbors_labels(X_train, y_train, x_new, k)
        y_pred.append(get_response_reg(neighbors))
    return y_pred
```

### 2.10 Testing

Should output an accuracy of 0.9473684210526315.

```
In [124]: # prepare data
          split = 0.67
          X_train, X_test, y_train, y_test = load_dataset(split)
          print('Training set: {0} samples'.format(X_train.shape[0]))
          print('Test set: {0} samples'.format(X_test.shape[0]))

          # generate predictions
          k = 3
          y_pred = predict(X_train, y_train, X_test, k)
          accuracy = compute_accuracy(y_pred, y_test)
          print('Accuracy = {0}'.format(accuracy))
```

```
Training set: 112 samples
Test set: 38 samples
Accuracy = 0.9736842105263158
```

### 2.11 k-Nearest Neighbors Problem Sets Solutions

```
In [125]: data = np.genfromtxt(fname='01_homework_dataset.csv', skip_header=1, delimiter=',')
          X_train, y_train = data[:, :3], data[:, 3]
          X_test = np.array([[4.1, -0.1, 2.2], [6.1, 0.4, 1.3]])
          k = 3
```

```
In [126]: y_pred = predict(X_train, y_train, X_test, k)
          print('Predictions:', y_pred)
```

```
Predictions: [1, 2]
```

**Problem 4**:

$$y_a = 1$$
$$y_b = 2$$

```
In [127]: y_pred = predict_reg(X_train, y_train, X_test, k)
          print('Predictions:', y_pred)

Predictions: [1.0, 1.3333333333333333]
```

**Problem 5**:

$$y_a = 1.0$$

$$y_b = 1.33$$

**Problem 6**: The means and standard deviations are of different magnitudes, specifically the mean of feature $x_1$ (0-indexed) is by 10e-2 lower than the other two. This resolves in scaling issues because the Euclidean distance takes all features into account.

```
In [130]: X_train

Out[130]: array([[ 5.5,  0.5,  4.5],
                 [ 7.4,  1.1,  3.6],
                 [ 5.9,  0.2,  3.4],
                 [ 9.9,  0.1,  0.8],
                 [ 6.9, -0.1,  0.6],
                 [ 6.8, -0.3,  5.1],
                 [ 4.1,  0.3,  5.1],
                 [ 1.3, -0.2,  1.8],
                 [ 4.5,  0.4,  2. ],
                 [ 0.5,  0. ,  2.3],
                 [ 5.9, -0.1,  4.4],
                 [ 9.3, -0.2,  3.2],
                 [ 1. ,  0.1,  2.8],
                 [ 0.4,  0.1,  4.3],
                 [ 2.7, -0.5,  4.2]])

In [132]: means, stds = np.mean(X_train, axis=0), np.std(X_train, axis=0)
          print(means, stds)

[ 4.80666667  0.09333333  3.20666667] [ 2.98741062  0.37321427  1.40592397]
```